CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Computer Science

Bachelor's Thesis

Framework for the use IoT sensor data in context-aware applications

Jan Svačina
Study program: Software Engineering and Technology

May 2018
Supervisor: Ing. Michal Trnka

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Jan Svačina

Studijní program: Softwarové inženýrství a technologie

Název tématu: Framework pro využití dat z IoT senzorů pro context-aware aplikace

Pokyny pro vypracování:

1. Proveďte rešerši existujících řešení pro zpracovaní dat z IoT senzorů a jejich využití pro context-aware aplikace.

2. Navrhněte architekturu frameworku, který by zjednodušil přístup k datům ze senzorů ve vyšším programovacím jazyce (např. Java). Zdokumentujte alternativy a důvody pro volbu dané architektury.

3. Implementuje knihovnu v Javě, která umožní zpracování dat z IoT senzorů. Použijte vhodnou metodiku vývoje SW, implementaci adekvátně zdokumentujte a otestujte.

4. Knihovnu demonstrujte na ukázkovém případě context-aware aplikace s několika různými senzory.

5. Zhodnoťte využitelnost dané knihovny pro obecnější případy použití.

Seznam odborné literatury:

1. Mohamed Ali Feki, Fahim Kawsar, Mathieu Boussard, and Lieven Trappeniers. 2013. The Internet of Things: The Next Technological Revolution. Computer 46, 2 (February 2013), 24-25.

2. M. Kranz, P. Holleis and A. Schmidt, ?Embedded Interaction: Interacting with the Internet of Things?in IEEE Internet Computing, vol. 14, no. 2, pp. 46-53, March-April 2010.

3. Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. 1999. Towards a Better Understanding of Context and Context-Awareness. InProceedings of the 1st international symposium on Handheld and Ubiquitous Compu-ting(HUC '99), Hans-Werner Gellersen (Ed.). Springer-Verlag, London, UK, UK, 304-307

Vedoucí: Ing. Michal Trnka

Platnost zadání do konce zimního semestru 2018/2019

V Praze dne 07.09.2017

Acknowledgement / Declaration

I thank Ing. Michal Trnka, the supervisor of the bachelor thesis for his valuable and always constructive advice and suggestions.

I declare, that i have done assigned the bachelor thesis alone led by supervisor. I used only literature, that is listed in work. Furthermore I declare, that I have no objections against lending or making public of my bachelor thesis or it's part with agreement of department.

In Prague 17. 5. 2018

.....

Abstrakt / Abstract

Bakalářská práce se zabývá vývojem efektivního frameworku umožňujícího programátorům vytvářet aplikace s vědomím kontextu, které zpracovávají data z IoT senzorů za předpokladu kratšího času pro vývoj, jednodušších programátorských konstrukcí a s využitím přístupu řízeného událostmi. Řešení umožňuje použití frameworku se širokou škálou senzorů, které posílají data samostatně, periodicky a ve vysoké frekvenci. Framework je postaven na technologii Spring Boot s použitím technik pro aspektové programování a přístupu řízeného událostmi. Integrace frameworku může probíhat s různými komunikačními technologiemi. Na základě zátěžových a statických testů je porovnáno konvenční a navrhované řešení. Všechny aspekty frameworku jsou otestovány a je dokázáno, že je efektivní v následujících doménách: kód je čitelnější, framework se časově vyrovná konvenčnímu řešení, kód je méně náchylný na chyby a vývojáři potřebují méně řádků kódu.

Klíčová slova: vědomí kontextu, IoT senzory, JAVA Framework, přístup řízený událostmi, reflexe

This bachelor thesis deals with the development of an effective framework allowing programmers to create context-aware applications using IoT sensor data, easier, in shorter period of time and with event driven approach. The designed solution allows framework to be used with wide range of sensors which send data simultaneously, periodically and with high frequency. Framework is built on Spring Boot using aspect oriented and event driven programming. It can be integrated with various communication technologies. The solution is compared with a conventional approach to developing context-aware applications by conducting stress and static tests. All framework's aspects are tested, and it is proven that framework is effective in following domains: it makes the code more readable, framework is competitive in time performance, code becomes less prone to make mistakes, developers need less amount of code.

Keywords: context-awereness, IoT sensors, JAVA Framework, event driven approach, reflection

Contents /

Tables / Figures

at11.0pt[10]

Chapter 1
Introduction

Computers had a clear dominance on the Internet by 2000, but from the turn of the century, other, such as mobile devices, began to connect to the network [1]. This trend continues and today we are talking about the " Internet of Things " [2]. Moreover, IoT should link 20.5 billion devices by 2020 [3].

Many of such devices can be assumed to communicate with each other, pass data, respond to each other, and use the context in which they are located. These devices are aware of the environment, so they are "context-aware " [4].

Transportation, housing, healthcare, and smart production will require an IoT network that is simple and reliable. The success indicator will be the TCO ("Total cost of Ownership"), which is how much the technology will cost from design to maintenance. In includes besides speed of product delivery, adaptation to new requirements and many more which relate to IT development [5].

Optimization and efficiency will play a vital role on all layers of the IoT model. Computing units will receive data from devices that will often be broadcasting at the same time without receiving authorization. The principle of "Connectivity as a Service" will drive the hardware and software development [6], i.e. the use of a network directly for business, such as detection and location tracking without the use of GPS locations.

Currently programmers are developing complex platforms [7] [8], which primarily allow communication among different devices, but no longer focus on optimizing programmed *business logic* for context-aware applications. It therefore creates different tools for converting different channels of communication and then processing is purely left on the programmer itself. The principle of context-aware is just a neglected factor in the construction of the IoT frameworks altogether.

This bachelor thesis goals are to investigate current solutions of developing context aware applications processing IoT sensor data, analyze and propose solution that would be TCO efficient, follow principle of connectivity as a service and support context-awareness and event driven approach. The framework should focus on the responding to events, executing chained subsequent events under certain conditions.

The thesis states fundamental terms to define the context of the work. Existing principles and methods with their usage for context-aware applications are described in the next chapter. By defining environment in which the framework will be used most often, the framework's requirements and architecture are described. The framework's architecture simplifying access to data from sensors is compared with alternative approaches and benefits and disadvantages are evaluated. Implementation is subject of the next chapter with subsequent testing and use case study, which proof framework's requirements fulfillment and show framework's capabilities with real-world devices.

Chapter 2
Background

2.1 Context Awareness

Situational information is commonly used in communication among people. Such information specifies the context in which members of information exchange are located. If we improve the access of computers to the context, the communication between man and computer will be enriched and thus allow to produce more usable products. Context is defined as *a*ny information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object [4]. Context-awareness or context-aware computing is defined *a*s the use of context to provide task-relevant information and/or services to a user [4]. Three important areas of context-awareness behavior are defined by [4]:

- presentation of information and services to the user
- automatic service execution
- marking context as information for further use

The main challenges in context-aware calculations are by [4] as follows:

- sorting, naming, and unifying types of contexts
- create infrastructure to support design, implementation, and development of context-aware applications
- discovering context-aware applications to help us interact with computing services

Creating infrastructure to support implementation of context-aware applications is subject of this work.

2.2 Context life cycle

According to [9] there are four essential steps in context life cycle. This life cycle is shown in the picture 2.1. Firstly, context needs to be defined, or gathered from sensors, next *n*ew context information needs to be defined in terms of attributes, characteristics, relationships with previously specified context, quality-of context attributes [9] and more. Defined context is used for deduction of new knowledge but inaccuracy in sensors must be validated against some historic data. Finally, the context information is presented to the users [9].

2.3 Internet of Things

The Internet of Things is the concept of sensors, mobile phones, tags, which can interact, *through unique addressing schemes, with each other and cooperate with their neighbors to reach common goals* (shorted) [2]. There are more visions of the IoT paradigm [2], the internet oriented vision assumes things to be connected via internet network. The

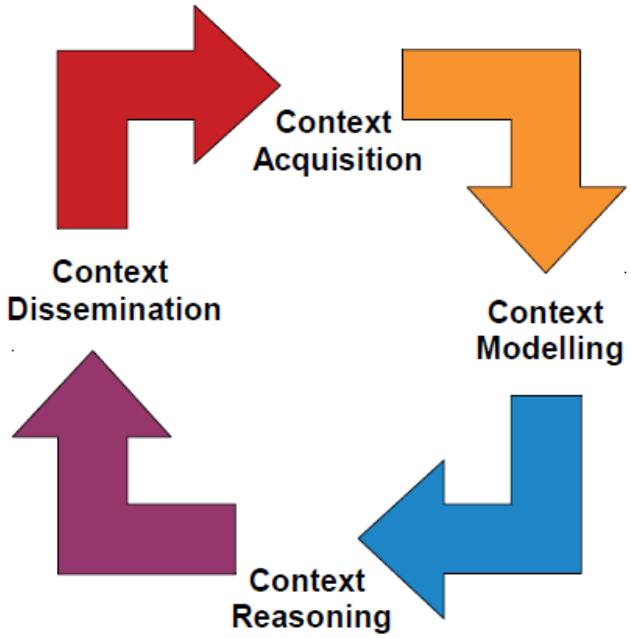

Figure 2.1. Context life cycle (Undertook from [9])

communication would be processed via servers, actuators which will be responsible for data and event management. Things and semantic oriented vision represents idea of uniquely addressed objects, with a direct targeted communication but storing and exchanging data became gradually a challenging issue [2] due to lack of computational and central storage capacity.

2.4 Micro-services

Context aware applications are designed in certain software architecture patterns. Micro-service is an approach to developing an application as a set of small services [10]. This pattern is suitable for using in IoT since it provides possibilities for M2M (machine to machine) communication. Services are deployed independently and provide certain functionality. These services are in the interaction with other applications and together they build system or systems with a desired functionality.

However, the communication schema can very depend on the requirements. There are few approaches presented in the [10]. Application can directly communicate with the services or via some message bus. The most interesting communication schema is a gateway as shown in the picture 2.2. This gateway could be represented by the application which serves only as a router with some additional filtering and invocation function. This schema draws first high-level overview of intermediate context-aware application which quickly evaluates, filters and invokes events based on incoming IoT sensor data from, for instance, components with a micro service architecture and (or) directly connected applications.

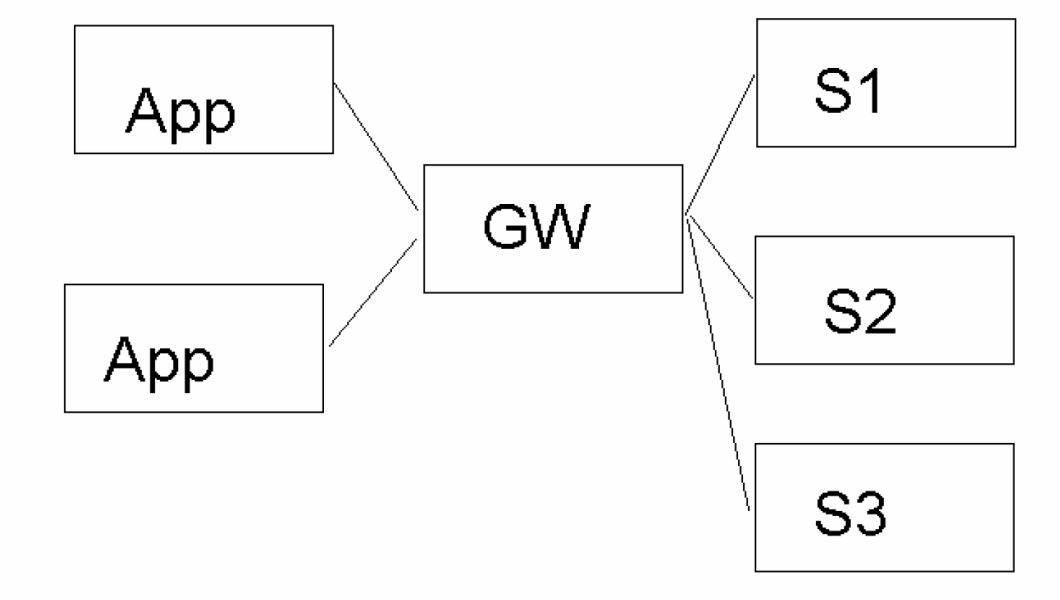

Figure 2.2. Micro service gateway (Undertook from [10])

2.5 Data management

Data management of any network must follow certain patterns to be manageable and scalable. According to [11] there are four main area in which data lifecycle happens. Application or Services are considered as actors which query data from the IoT system. The querying data can be historical or global and is processed from offline backend which is storage-centric, typically data persisted in some databases. Querying can be also real-time or localized directly to autonomous devices, IoT sensors. These sensors process and send data back or can transmit data to online frontend, communication centric system. The frontend collects, filters and fuses measured data and then push the data either to backend offline system or to applications. It is communication centric, thus it highly interacts with the user or the outside environment. The schema 2.3 provides overall picture of modelling IoT data lifecycle.

Context-aware applications query data form sensors and expect either quick response or for instance regular updates pushed by frontend system. Such applications might access data in all states represented in the schema.

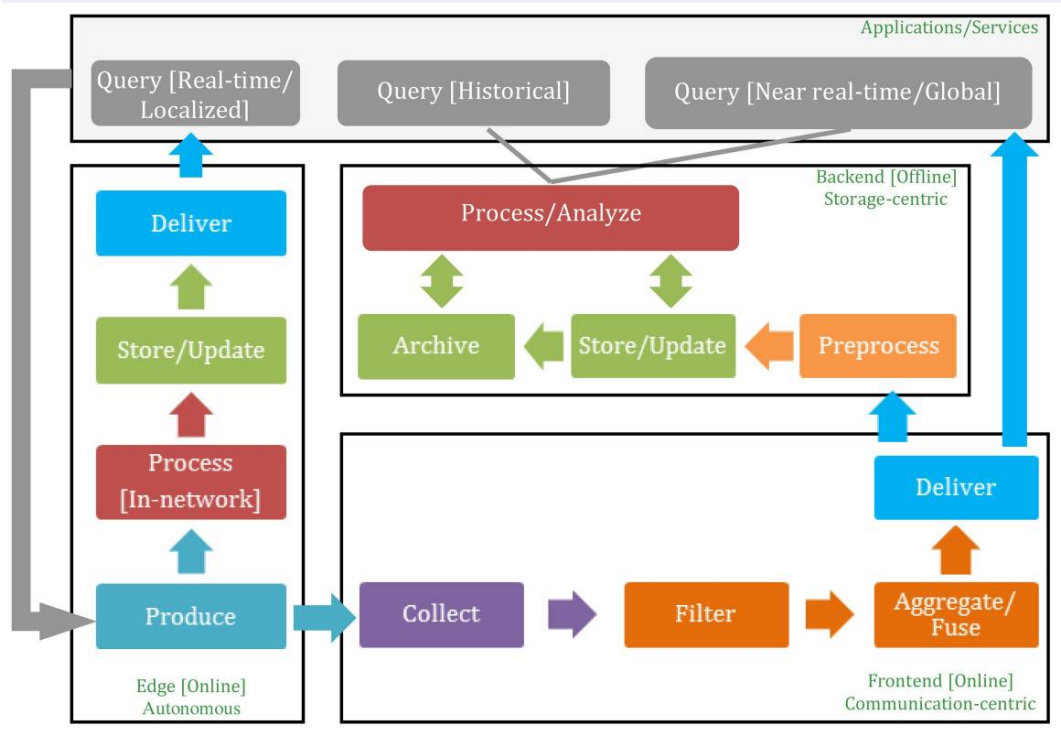

Figure 2.3. Data management schema (Undertook from [11])

Chapter 3
Related work

3.1 Conceptual framework of context-aware applications

The conceptual framework presented in [12] deals with handling the context in applications. Authors introduce several components with distinct function which together process data regarding context. *C*ontext widgets acquire context information. Interpreters transform and raise the level of abstraction of context information. Interpreters may combine multiple pieces of context to produce higher level context information. Aggregators gather context information related to an entity for easy access by applications. Services execute behaviors on the environment using acquired context. Finally, discoverers allow applications (and other components) to determine the capabilities of the environment and to take advantage of them [12].

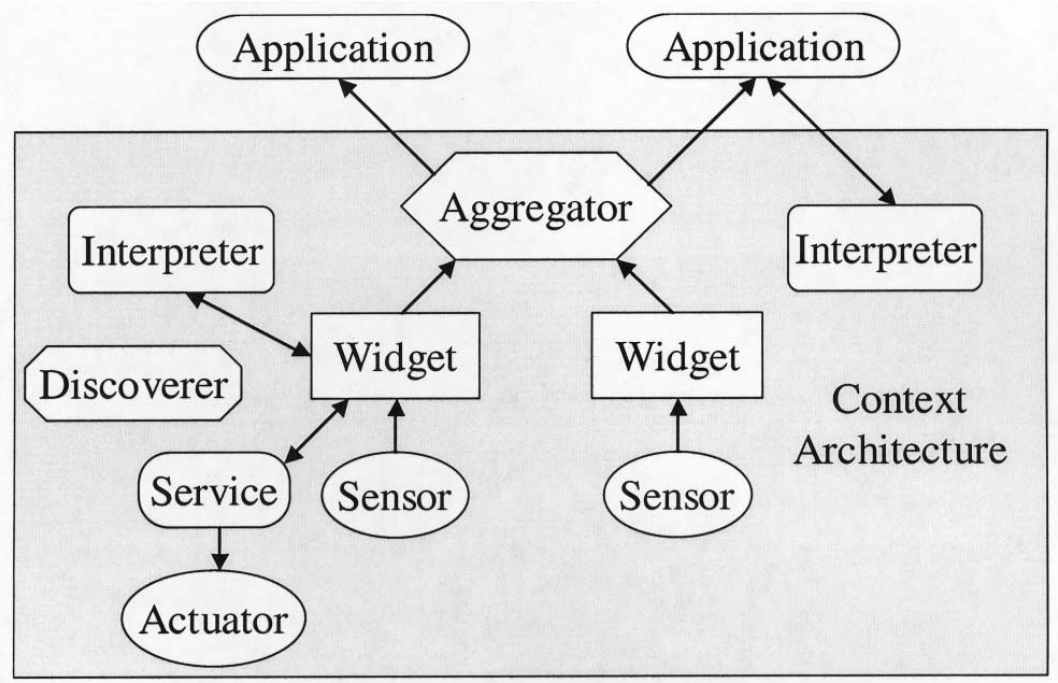

Figure 3.1. Conceptual framework of context-aware applications (Undertook from [12])

This architecture again shows the need for some server application that would effectively implement some business logic of invoking and filtering requests. It introduces an aggregator, which is an interface for applications, handles the communication and makes sensors as shown in figure 3.1. This aggregator stores information about the entity and might collect data gathered form sensors.

3.2 Data management framework

Data processing from IoT sensors is divided into two parts by [11]. The first is the so-called online layer that intensively communicates directly with connected devices, broadcasts and receives requests in high frequency. Second, the offline layer then performs a deeper analysis of the data and saves it for later processing. Both layers communicate intensively with each other, providing data to each other and evaluating the status of connected devices.

However, this reasoning does not tell us too much about real data processing, rather it shows the concept of data processing in a high degree of abstraction, which covers technology from sensor devices to the cloud [11]. This is a logical abstraction for splitting a data processing system that is more related to hardware and that is more associated with the application layer.

The proposed data management framework for IoT by [11] defines a layer for each IoT data state from consumption of requirements in application to collecting data from sensors. The model aims to connect all systems, transportation, medical care, environmental monitoring via different networks (LTE, GSM). It emerges storing, querying and management of the data. It aims to create model for global IoT data management to make joining of new technologies and paradigms easier.

3.3 Simurgh framework

Internet of Things includes variety of services from more hardware oriented to the cloud computing. Simurgh framework [13] comes with an architecture that tries to provide desired platform for functionality provided to end users. The figure 3.2 shows three self describing layers, end user, platform and things layers. The Platform layer deals with the flow, user is requesting some services, which use sensors on the things layer. Request propagates through API Mediator and others to finally provide the requested data or functionality.

Flow composition is one of platform layers and is important for this work. Authors described this layer as a *component capable of combining two or more flows to build a new flow that delivers new functionalities* [13]. This layer is thus responsible for event management and thus the business logic of the application is most likely to place here. Other layers provide necessary communication APIs, but the flow composition layer is crucial part of the software from the business view. The IoT framework, which is subject of this work, would be exactly deployed in this layer to provide the desired flow.

3.4 Context-aware applications using IoT sensor data

In Embedded Interaction [14], authors write about the usability of data collected from objects that we all know and use. They used kitchen utensils, clothing or sports equipment. Such things were attached by the simple sensors that transmitted data about their usage. In the case of the kitchen countertop were used data of the pressure on the plate, the method of slicing, the weight of the items placed on the board, etc. Other examples that are listed in the publication are sensors placed on clothing and toys. The sensors then served to play educational games with a sports theme.

The last use case was the location of the sensors on the office door. According to the office owner's calendar and his current schedule sensor detects, whether the user is at work, of he comes in a moment or has already gone home and is closed.

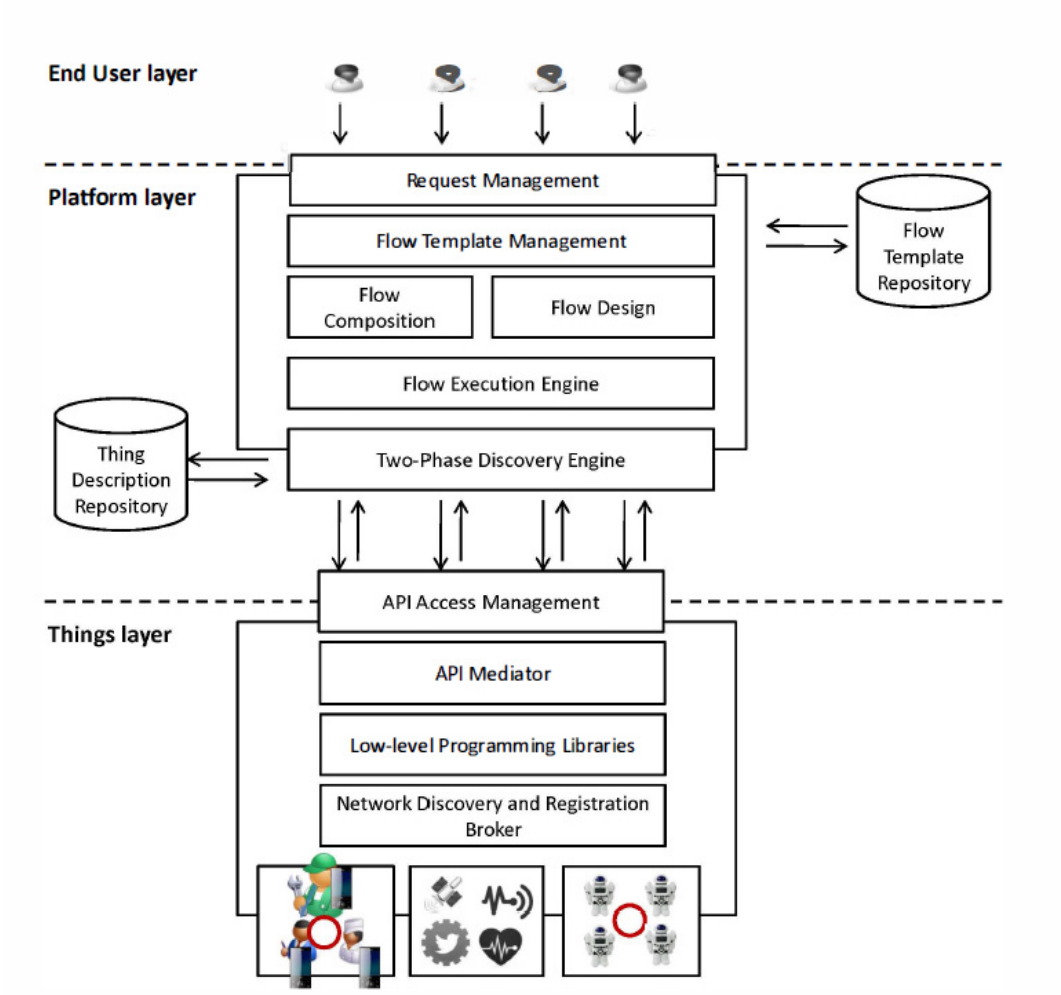

Figure 3.2. Simurgh framework overview (Undertook from [13])

Data obtained from sensors is typically used primarily to optimize already existing user activities or to offer the use of new ones. The sensor on the door of the office allows user to discard worries about the correct placement of the work status card. In contrast, the data from the kitchen offers the possibility to show user improvement the cutting technique of certain types of vegetables or offers the possibility of sending ideas about healthier the diet. These possibilities can be extended quickly to various use cases from daily life.

In terms of user interaction with the system is important to distinguish how is user aware of the system. Explicit interaction means that the user controls the device completely consciously to achieve some goal. Implicit means that the user focuses primarily on his goal, using tools is aware, but his communication with the computer system is not recognized [14].

3.5 Toolkits

Programming context-aware applications using IoT data requires advanced architectures as shown in previous sections. Kranz and his group decided to use so called EIToolkit [14]. It is a kit that has component-oriented architecture, which enables communication with connected devices or to spread messages.

Such architecture makes it very difficult to program effective logic across sensors and use cases. A layer is needed to increase user-friendliness of applications, to link sensors across use cases and to provide much broader possibilities in business logic programming.

Kranz case study in the kitchen, his educational game and office equipment would surely have gone to connect each other and increase context-awareness of the applications. The authors themselves state the possibility of having a context as crucial for the entire IoT concept.

Kranz and col. wrote about contextual dependencies: *The value of having access to information depends on context. Many different contexts, make a whole range of sensors and input processing systems necessary. For Most context-aware applications, focusing on just a person, an object, or a specific environment is meet meaningful, but in the Internet of things, these borders merge and vanish* (shorted) [14]. Component architecture, however, appears to rather limit the context.

Chapter 4
Design

Modern applications using context in combination with sensor data have the potential in many sectors of industry and everyday life. The context will be key to the quality of the application and will determine the further development and direction of applications. Certain application types have the same functionality elements that can be delegated to external libraries or frameworks. The IoT Framework would be used by programmers for a software development, thus it would become sort of product and would be integrated into the existing projects.

Expectations are put not only on the qualities of the framework as such, but also on its behavior in the environment into which it is substituted. The developer uses additional libraries and custom procedures. It is expected that such framework will not affect the operations of other modules and developer would not have to modify existing programming techniques.

The application requirements depend on the usage of sensors data for context aware applications. A typical example is an event that triggers additional events when a certain condition is met. An example can be a car that arrives in a garage in a building. Arrival is recorded in the system, which then starts the air conditioning, if the temperature is too high, the lights in the building will light up if it is dark and closes the driveway when there are no obstacles in the way. Another example is the usage in industry. Automatic notification is sent when inventory is required, followed by the subsequent execution of replenishment processes. Public services in crisis situations are other prime examples. Usage is obvious everywhere where X is the initiator of several different processes running under certain conditions. Triggered event can also invoke another process in the chain reactions.

4.1 Chain event invoking

Invoking an event causes asynchronous call of other events, which triggers another chain of events, etc. The event, which starts the chain of event triggering let be origin event and the events invoked by the origin event let be subsequent events. Subsequent event can be also origin event for another set of subsequent events. Invoking origin event than spreads method invocation through whole application. It is the case that dependencies might be set to the circle. This behavior is not only allowed but sometimes also wanted when events need to loop over time meet certain conditions.

4.2 Condition event filtering

A solid asynchronous event invoking is powerful tool but indeed needs to be completed by functionality which allows developer to restrict event invoking under certain conditions. From the example above, event is invoked only when temperature is risen over X. In this case, a value entering the method as an argument needs to be evaluated against certain static value defined by developer. Such restrictions also enables to stop event looping in circle dependencies as mentioned in previous section.

4.3 Qualitative attributes

Framework should have qualities in terms of design, thus enabling its easy usage in the various use cases and provide straightforward testing of programmed logic. Next, framework should be competitive in terms of performance and scalability, thus providing options to use framework in critical software. Moreover, framework should also provide clear way of using its functionality smoothly and orderly.

Design qualities

- Integrity: The naming conventions must be short and concise to match the logic that is being performed and, at the same time, its naming does not make the code invasive and distracting.
- Reusability: Framework logic must allow to use a variety of applications in sensor interaction programming.
- Adaptability: The programmer using the available functionality must be able to adapt the functionality to its specific needs in the development of SW.
- Compatibility: The application implementing the IoT Framework would interact with all sorts of systems that differ in complexity and requirements. Unified interface needs to be defined that allows to access functionality from different devices.
- Testability: The Framework's behavior will enable to conduct developer and load tests to eliminate developer errors and evaluate the time consumption of method calls.

Run-time qualities

- Performance: The Framework must have minimal overhead for the time it takes to execute its logic. This logic should be as high as possible in advance when the application starts.
- Scalability: The Framework must allow easy operating with devices in the quantity of pieces, as well as quantities over hundreds of devices.

User qualities

- Usability: Method names must be self-describing and their locations conform to the conventions used in the Java programming language and its built-in frameworks, for instance Spring Boot.

4.4 Architecture

Based on quality attributes in previous section, multi-layered architecture with *event-driven* elements was chosen architecture for the framework. It provides easy-to-use information exchange among applications and further advantages of multi-layered architecture goes as follows:

- Abstraction: Abstracts the system into several units so that it is easy to recognize the functionality of each layer and the relationships between them
- Encapsulation: Each layer contains its methods and variables, thus they are not reused in multiple locations of the system
- High Cohesion: well-defined functionality for each layer ensures the execution of one type of tasks across the layer
- Loose Coupling: Individual layers communicate with each other using clearly defined contact surfaces

4.4.1 Analytic model

Figure 4.1 captures packages and classes within framework with regards to relationships among them. It shows also frameworks interconnection with server application. Diagram shows that individual frameworks component are highly linked with each other. However, server application uses only annotations and model, thus it provides very clear way of using the framework.

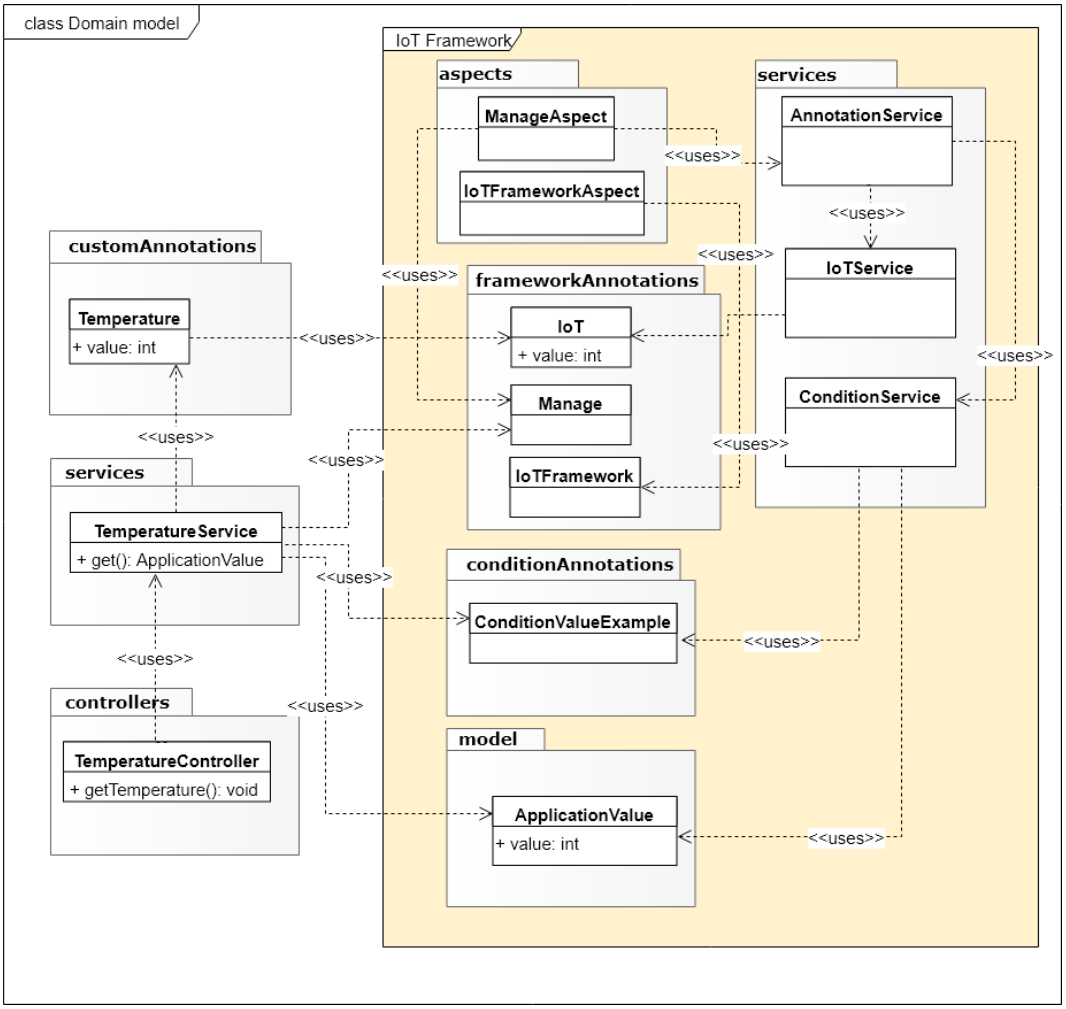

Figure 4.1. Analytic model of classes

4.4.2 Framework invocation life cycle

The Framework Lifetime Sequence Diagram 4.2 captures a user of a server application implementing the IoT Framework. Application starts up and by the end of booting it scans application for annotations and methods implementing the framework. These data are stored in inner cache of the framework for further usage. Next, user sends request to TemperatureController, for instance sending current data from thermometer. Controller triggers origin event defined in the TemperatureService, which from users point of view asynchronously invokes subsequent events. Meanwhile framework performs following tasks. The origin event method call is intercepted by the aspect, which passes metadata about the origin event to AnnotationService. This service let the ConditionService to evaluate origin event input data with values defined in conditions. Positively evaluated events are invoked as expected.

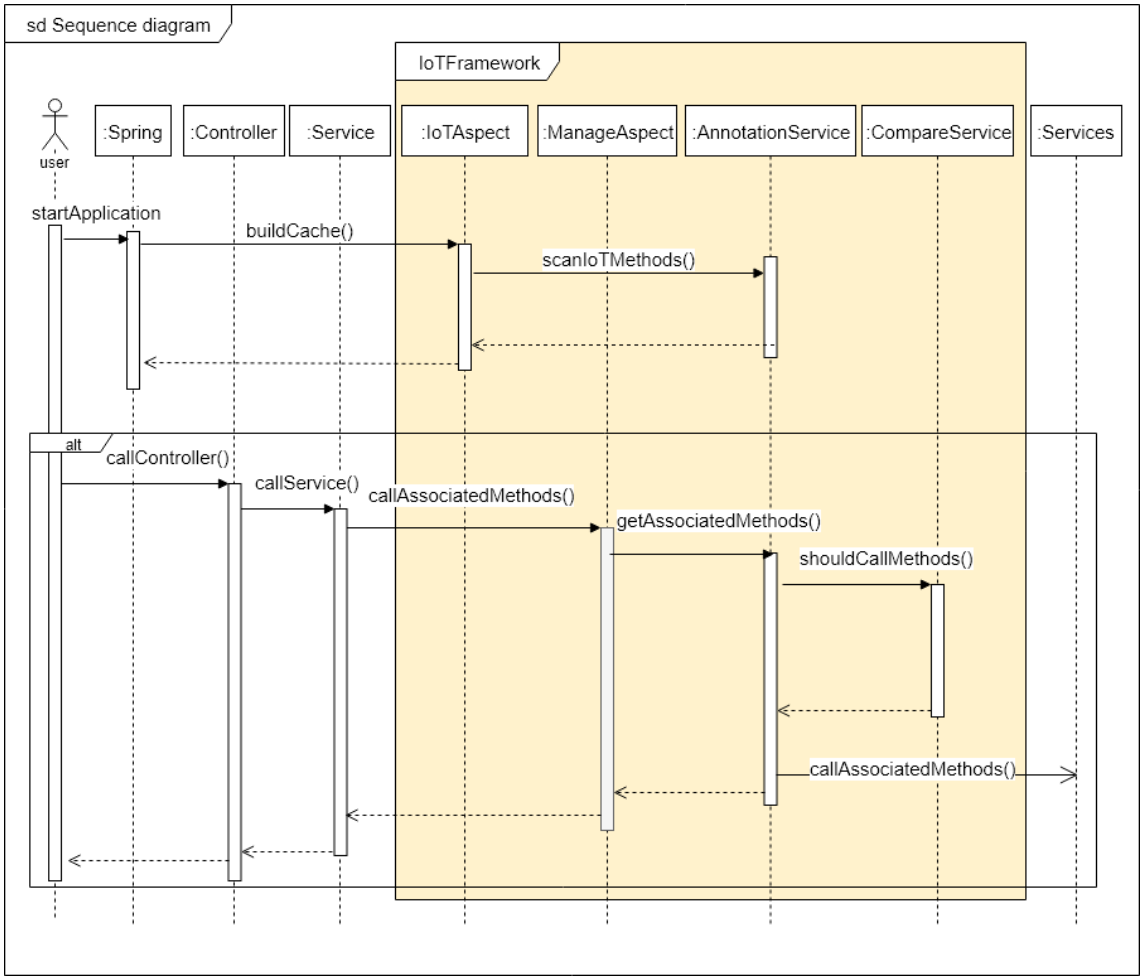

Figure 4.2. Framework Invocation Life Cycle Sequence Diagram

4.5 Architecture alternatives

Chapter 3 includes many examples of architectures, multitier, microservice, component or event driven architectures. The IoT Framework uses multi-layered architecture with *event-driven* elements. The benefits of such architecture are abstraction, encapsulation, high cohesion and loose coupling. The event driven elements accomplish reactions to IoT communication.

The architecture alternatives has one or more disadvantages that prevent its usage. Component oriented architecture defines approach of loosely coupled components, they are independent and changing individual components is easy for developers. This architecture also enables event handling thus being suitable for event-driven architecture. Pure component architecture needs to communicate together via interfaces, thus quick exchange of information through system might become complex. Component approach was partly used via aspects as described in the chapter 5, because this approach decreases the amount of code compared to classical solution with abstraction of builders, abstract classes and interfaces.

Monolithic architecture provides easy communication within the system, but the complexity of the framework requires more advanced solution, which would be suitable for future scaling and development.

Multitier architecture is from all alternatives very close to the desired architecture. Provides abstraction and encapsulation, such system can be easily extended and manageable. This architecture was used in Simurgh framework and Data management framework presented in the chapter 3. The event driven elements, however, are needed to use advantage of IoT. Sensors are changing their state, consumes incoming data at high pace. Event driven architecture can transmit then the events through loosely coupled services, components or layers, therefore the multilayer architecture with event driven elements was used.

4.6 Technologies

Framework will be used in higher programming language JAVA and thus the recent features can be used to enhance frameworks capabilities in order to provide wide range of benefits for programmers.

This can be ensure only by picking up the most cutting edge technologies and patterns which have been gradually adopted by the community over the years.

4.6.1 Spring Boot

An open source framework for creating Java Enterprise applications. The advantage of the framework lies in the already configured tools for MVC and AOP. Java was selected because of its support for OOP, AOP, and it is independent of the platform and one of the most widely used programming languages. Spring was used instead of JEE in terms of greater simplicity and because it can be executable on any Java platform [15].

4.6.2 Spring AOP

Aspect oriented approach separates common use cases across application, such as logging or security, resulting in cleaner code. Spring library for aspect oriented programming uses AspectJ library, which provides aspects, joinpoints, advices and pointcuts.

Aspect is a class that can modify code behavior via advice. Arguments, name of methods or classes of non-aspect code are accessible and modifiable in the run-time of the system. Different places in the code (joinpoints) are targeted via pointcuts, which are expressions incorporating set of methods, classes or annotations. Targeting annotations with pointcuts appears to be extremely powerful since these annotations, syntactic meta data, are lightweight and code remains clear when using annotations in different places in the system [16].

This functionality is precisely used in the framework since event listener in form of pointcut is bind to execution of all methods with a @Manage annotation. As the method is invoked, framework catches the event and execute its own logic of calling subsequent methods with managed annotation.

4.6.3 JAVA Reflection

Reflection is generally ability of a programming language to find out data and syntactic structure of some object in the run-time. To name few, annotations or attributes of clases, methods, enums and another objects is possible to retrieve by reflection. The reflection happens in certain package or packages.

The very first intended JAVA reflection usage in the IoT Framework is to scan application to retrieve all methods annotated with certain IoT annotation. This can happen on the run-time of the program, but results in delays. The solution is to introduce cache as shown in the figure 4.1. Links among iot and manage annotation shall be established

on the start up of the application since new methods and annotations are not added in the run-time. Framework can access data in the cache at any time in the run-time of the program.

4.6.4 Maven

Dependency management tool Maven is used by a large community of developers. More experienced programmers are accustomed to it. Public repositories contain a large database of dependencies [17].

The IoT framework was placed to the public maven repository to be easily accessible by the community.

4.6.5 Tomcat

Tomcat is a web server and Servlet container in Java. It is the most widespread Web server for Java [18] and therefore was used for developing and deploying the IoT Framework.

4.7 Comparison of approaches

Conventional and IoT Framework approach will be described in this chapter on an use case of the server application receiving data form large amount of sensors. Besides, complicated logic of conditions and reading data form sensors will be presented. I will show you how developer would code given use case using conventional approach and than using smart approach via IoT Framework.

A factory is prime example, where could be found tough conditions for humans and machines. Material and products is subject of many government restriction and thus quality assurance is one of the most crucial requirements with noticeable impact on business itself. This is issue not only for employers bud also employees to let sensors track goods and environment properly. These conditions are measured by temperature, humidity and light sensors. Excising humans or devices to higher temperatures or humidity or intense light leads to permanent damages or health issues.

Following sensors are supposed:

- temperature
- humidity
- motion
- sound

All sensors are suited on various places in quantity of tens or few hundreds.

Following use case is considered. Cast iron product is in the forge, worker is exposed by high temperatures and low humidity. If temperature rises above X, check if humidity decreased under value Y, turn the warning diode on, start Z time units interval, if the worker has not left the place in Z time units, turn on sound, stop interval if the time exceeds 5Z time units, notify senior staff if the interval exceeds 2Z. If the humidity decreased notify quality assurance and if decreased two times, stop the machine.

4.7.1 Logical flow

Such complex use case is programmed absolutely neatly with the framework within few minutes. Programmer just writes down the list of methods as shown on the picture 4.3

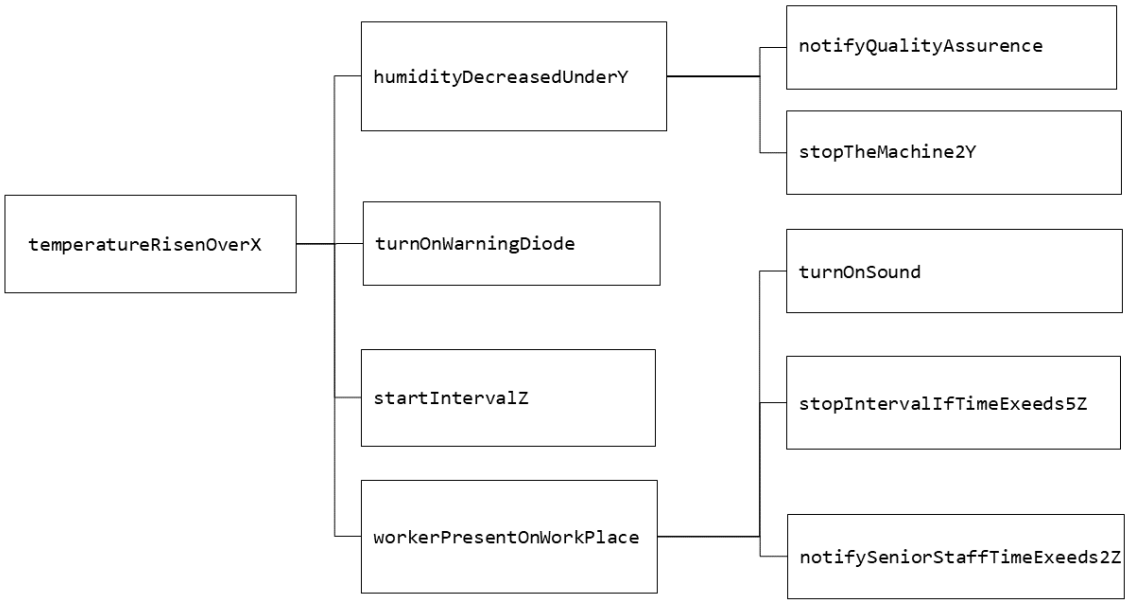

Figure 4.3. Use Case Logical Flow

4.7.2 Adding conditions

Methods are annotated with condition annotations and the ConditionValue is put as an argument to the condition annotation. Method's argument with value from the sensor will be evaluated against value in the annotation. This erases all complicated if statements and debugging the pieces of code which are prone to mistakes as shown in figure 4.4.

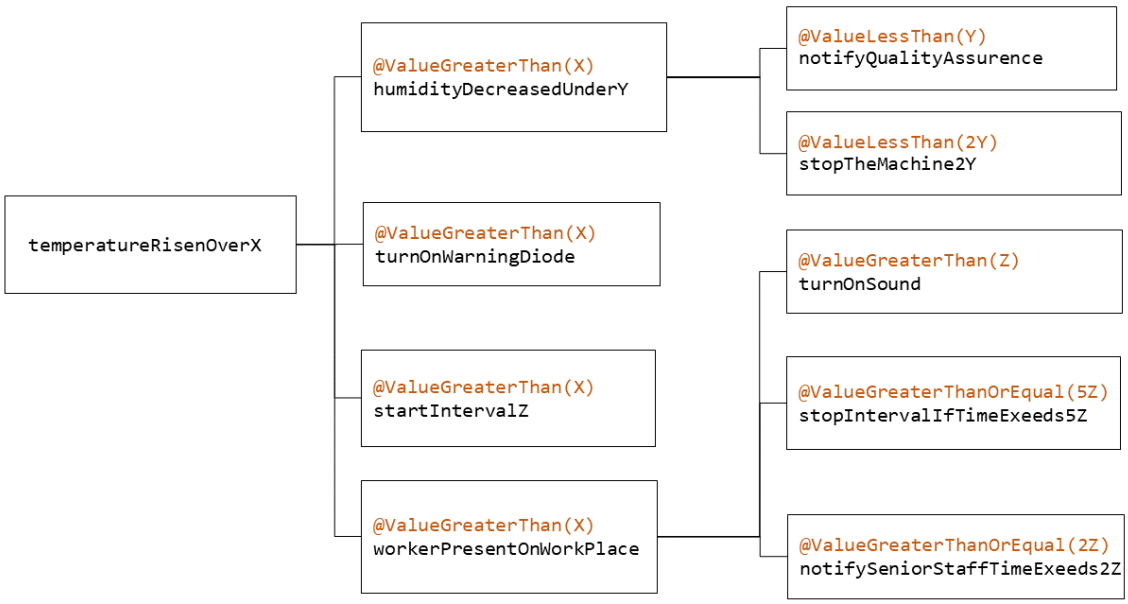

Figure 4.4. Adding Conditional Annotations

4.7.3 Building hierarchy

Finally, developer annotates methods with invoking annotations for managing origin and subsequent events logic by defining his own annotations and by adding manage annotation. This is clear and simple setup of event hierarchy as shown in figure 4.5.

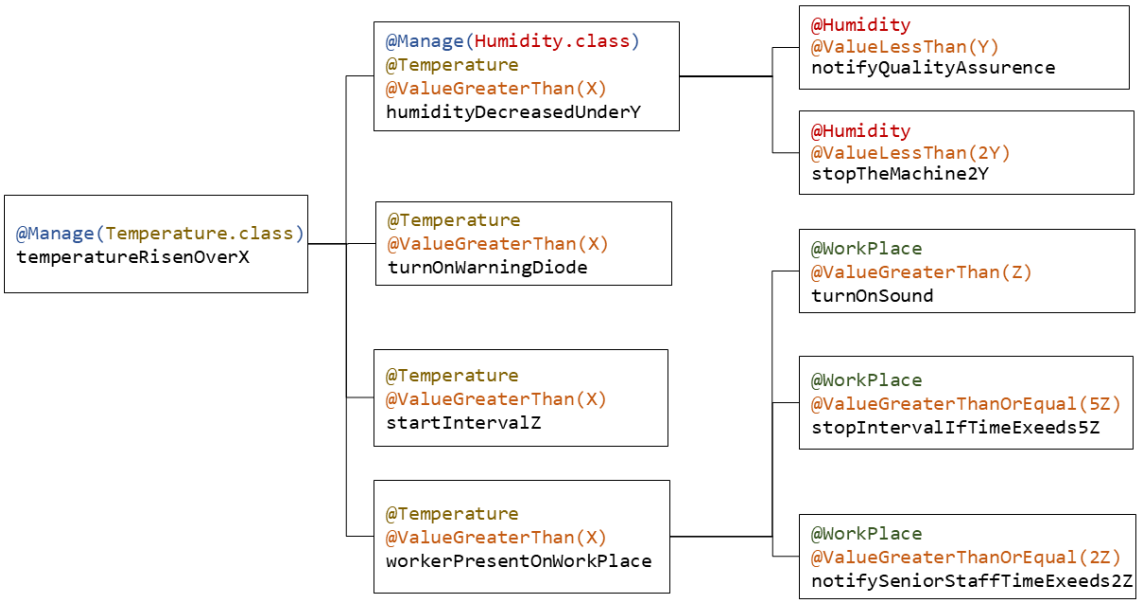

Figure 4.5. Adding Invoking Annotations

4.7.4 Code comparison

Framework's simplicity is observed on another example showing two snippets of code in figure 4.6. The mentioned snippets shows code of TemperatureService in the above example of use case in factory. On the left there is code using standard procedures of autowiring, or injecting beans and on the righ there is example of code using IoT Framework. At first sight there is a significant difference in amount of code, structure and its clarity.

```
@Autowired
private HumidityService humidityService;

@Autowired
private WorkPlaceService workPlaceService;

public void temperatureRisenOverX(int temperature){
    humidityService.humidityDecreasedUnderY();
    workPlaceService.workerPresentOnWorkPlaceInZ();
    if (temperature > X) {
        workPlaceService.turnOnWarningDiode();
        workPlaceService.startIntervalZ();
    }
}

public void turnOnWarningDiode(){}

public void startIntervalZ(){}
```

```
@Manage(Temperature.class)
public void temperatureRisenOverX(){}

@Temperature
@ValueGreaterThan(X)
public void turnOnWarningDiode(){}

@Temperature
@ValueGreaterThan(X)
public void startIntervalZ(){}
```

Figure 4.6. Comparison of IoT Framework and conventional approach

Chapter 5
Implementation

Chapter 4 stated overall technologies, which should be used in combination with the framework. However, IoT framework itself requires more advanced technologies in order to fulfill all subsequent requirements in the mentioned chapter. Firstly, author decided to implement event driven elements via aspect oriented approach. This approach defines a set of paradigms in which developer can easily add listeners on various events, mainly on executing methods with some characteristics which distinguish them. Aspect example is shown in listing 5.5. Defining listener, so called `pointcut` is shown in listing 5.4 Another used approach is reflection, which provides handy functionality for scanning through application or its part. Reflection is able to retrieve set of class or methods based on some common characteristic, for instance being annotated by certain annotation as shown in listing 5.9.

5.1 Frameworks structure

Frameworks structure consists of Annotations used for method management, aspects for triggering events, services holding the frameworks cache and Application Value for holding input data as shown on figure 5.1. Framework structure was design with regard to its easy future development. All folders will be described in the following chapters with special regard to its functionality, usage for developer and impact on the system.

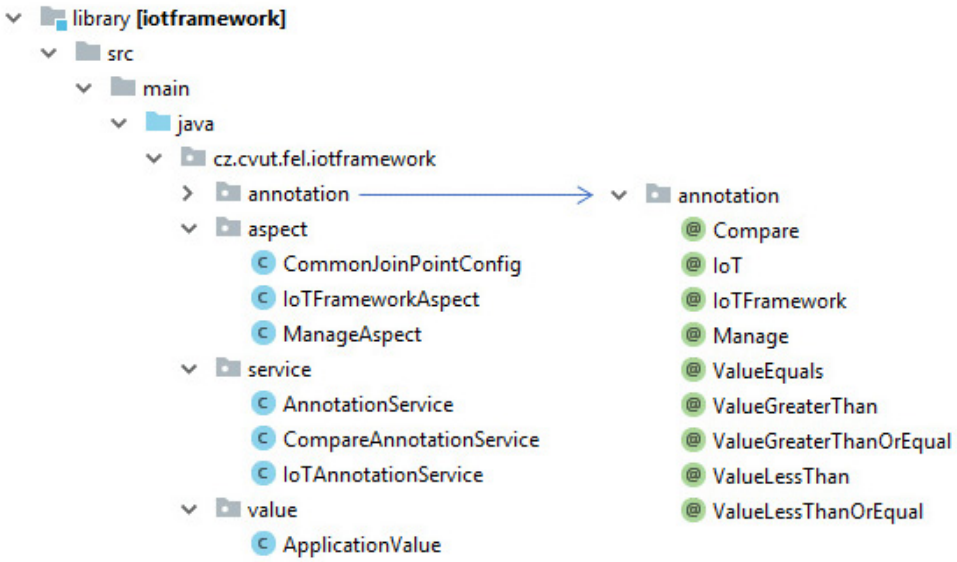

Figure 5.1. Frameworks directory structure

Annotation	Description	Arguments	Has Aspect
`@IoTFramework`	Creates cache	Folder paths	Yes
`@IoT`	Creates manageable annotation	None	No
`@Manage`	Invokes methods	Managed annotation	Yes
`@ValueEquals`	Creates condition	Condition value	No

Table 5.1. Description of framework annotations

Framework defines a set of annotations, which can be divided into three groups by their usage across the application. Table 5.1 shows its division with regard to the arguments they receive and whether they are used in combination with an aspect.

5.2 Chain Event Invoking

In chapter 4 was stated the requirement for chain event invoking and was explained its need. In this section data structures and programmed paradigms will be described in more detail. Annotations are used in JAVA form version 5 and it provides programmer option to create his own metadata tags, which can be placed on classes, methods or annother annotations. Annotations are introduced with key word `@interface`, than is defined target, on which elements annotation can be placed and retention policy, under which circumstances of application life time the annotation will be active. In the listing 5.1 is shown annotation `@IoT` which is part of the framework and is used by the developer to annotated his own annotations, which will used to provide asynchronous evenet invoking.

```
@Target(value = {ElementType.ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface IoT {

}
```

Listing 5.1. IoT Annotation

The listing 5.2 shows how developer creates annotation which will be used for creating cache. Target is changed to method and retention policy remains same. Such created annotations implementing `@IoT` shall be know for simplicity as IoT annotations.

```
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@IoT(value = 0)
public @interface Temperature {

}
```

Listing 5.2. Temperature Annotation

Annotation shown in listing 5.3 has field value of type `Class<?>`. In this field shall be annotation which will be managed, i.e. all methods with the given annotation will invoked asynchronously. For instance IoT annotated annotation Temperature from listing 5.2.

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Manage {
```

```
    Class<?> value();
}
```

Listing 5.3. Manage Annotation

In order to target add listener on `@Manage` annotation, a pointcut must be created as shown in listing 5.4. This pointcut is only an selector which will be used further as shown in listing 5.5.

```
@Pointcut("@annotation(cz.cvut.fel.iotframework.annotation.Manage)")
    public void manageAnnotation(){}
```

Listing 5.4. Manage Pointcut Showcase

Manage aspect shown in listing 5.5 is triggered when ever condition from pointcut is met. Via Spring AOP annotation `@Around` is invoking of method with manage annotation caught and following procedures are conducted. Application Value, the argument value of the method, and method annotations are passed to AnnotationService which performs tasks to call services annotated with managed annotation.

```
@Aspect
public class ManageAspect {

@Autowired
private AnnotationService annotationService;

@Around(value="CommonJoinPointConfig.manageAnnotation()")
public void manageAspectMethod(ProceedingJoinPoint joinPoint) {

joinPoint.proceed();

annotationService.callMethods(annotations,applicationValue);
}
}
```

Listing 5.5. Manage Aspect Showcase

In listing 5.6 is shown usage of actual frameworks annotation `@Manage` and by developer created IoT annotation `@Temperature`.

```
@Manage(Temperature.class)
public void processTemperatureData( ApplicationValue temperature) {
//...
}
```

Listing 5.6. Manage Annotation Showcase

From the terminology point of view, we can devided two events. There is the origin event starting the chain invocation cycle and subsequent event as shown in figures 5.2 and 5.3. To clarify the terminology used in this work, there are descriptions to all parts of the code.

5.3 Condition Event Filtering

In previous section was described the implementation of asynchronous event invoking. This section deals with an clear requirement, to restrict invoking event based on certain condition. Example of condition event filtering is shown in listing 5.7. The annotation's name express condition and value in the argument of the annotation represents right side of the equation. This value let be ConditionValue.

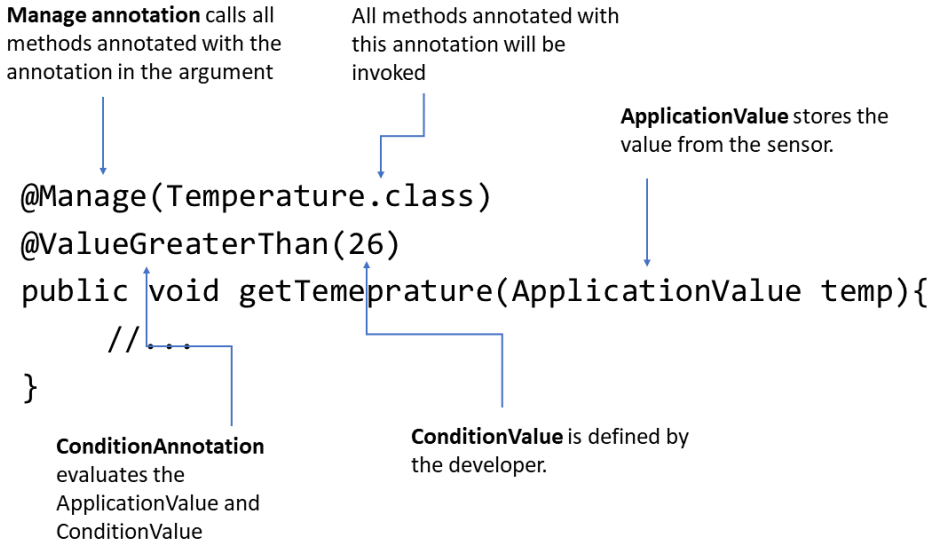

Figure 5.2. Origin event

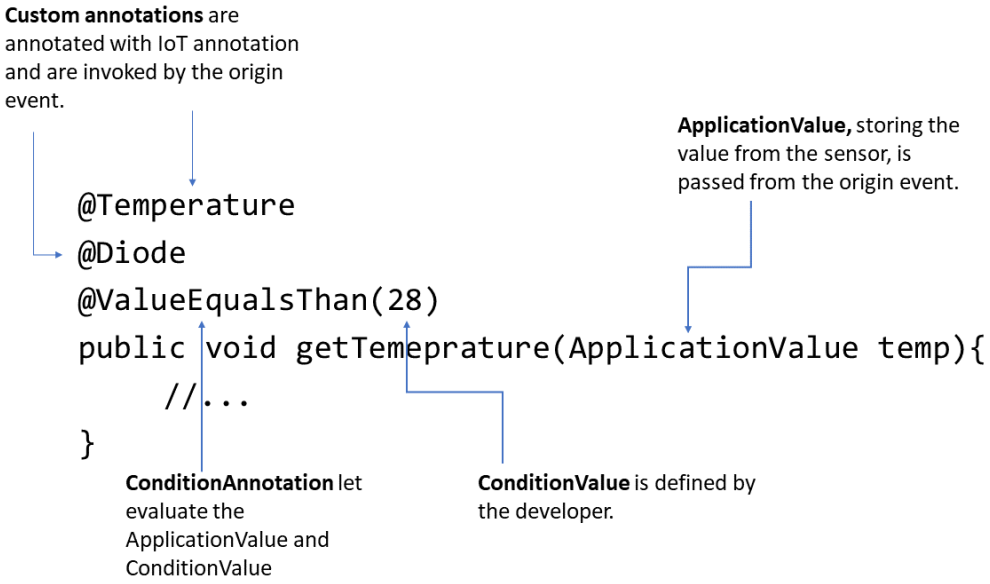

Figure 5.3. Subsequent event

5.3.1 ApplicationValue

The `ApplicationValue` is framework's class representing unified object with value from sensor, which will be compared against ConditionValue, thus providing desired functionality of condition event filtering.

```
@Temperature
@ValueGreaterThanOrEqual(26)
public void sendNotification(ApplicationValue temperature){
// ...
}
```

Listing 5.7. Condition Annotation

5.3.2 Example

Let be the ConditionValue 32. The equation in listing 5.7 would look this way: `32 >= 26`. The condition is met and therefore the execution of the `sendNotification` method can proceed. Expected functionality of ConditionValue and ApplicationValue can be described in the case of of `@Manage` annotation, for instance back in listing 5.6. The method `processTemperatureData` carries `@Manage` annotation with argument of `Temperature.class`, therefore it calls all methods with `@Temperature` annotation which will receive ApplicationValue from the `@Manage` annotated method and will evaluate the value against ConditionValue. Further case, worth to mention, is the same called method but managing another annotation. The ApplicationValue remains the very same from the origin event.

Overview all condition annotations is shown in the table 5.2. Naming convention is straightforward and indeed provides clear usage of desired functionality. Further development of complicated conditions might be subject of further development.

Annotation	Sign
`@ValueEquals`	`==`
`@ValueGreaterThan`	`>`
`@ValueGreaterThanOrEqual`	`>=`
`@ValueLessThan`	`<`
`@ValueLessThanOrEqual`	`<=`

Table 5.2. Condition Annotation Overview

5.4 Framework Cache

The `@IoTFramework` annotation provides functionality for initialization of the framework. It defines where annotations and services are stored and used within framework as shown in 5.8. This is essential for reflection as it scans only following packages and whole application which might grow over time. Secondly such approach force developers to follow design principle of separation of concerns. Annotation is recommended to bind together with the method triggered when application startup finished. Aspect is bind on the annotation and starts caching via reflection in the two folders, which path was provided in the annotation definition.

```
@Override
@IoTFramework(
services = "cz.cvut.fel.iotserver.service",
annotations = "cz.cvut.fel.iotserver.annotation"
)
public void onStartup(ServletContext servletContext)
throws ServletException {
}
```

Listing 5.8. IoT Framework Annotation

Scanning the application provides reflection, which is able to retrive set of methods or classes with some characteristics. In the listing 5.9 is scanned package with IoT annotations created by the developer and used for asynchronous event invoking.

```
public Set<Class<?>> setAnnotations(String annotationPackage){

Reflections reflections = new Reflections(new ConfigurationBuilder()
.setUrls( ClasspathHelper.forPackage(annotationPackage) )
.setScanners( new TypeAnnotationsScanner() ) );
return reflections.getTypesAnnotatedWith(IoT.class);
}
```

Listing 5.9. Reflection Example

```
public HashMap<Class, Set<Method>> annotationMethodCache
= new HashMap<>();

public HashMap<Method, Integer> methodIntegerReferenceCache
= new HashMap<>();

public HashMap<Integer, Object> integerObjectCache
= new HashMap<>();
```

Listing 5.10. Creating Frameworks Inner Cache

The following listing 5.10 shows framework's cache in the `AnnotationService`, which stores links between annotations, classes and methods in the following relationship. Each IoT annotation has link to set of methods that carry the actual annotation. Each method is reference to the object on which is executed.

5.5 Limitations

Framework might bring less benefits when using in small applications with few sensors and simple use cases. This approaches would focus more on simplicity and thus framework would become unnecessary.

The framework also scans application in the defined packages with annotations and methods implementing the framework. This limitation might be discouraging for developers since the logic of the application might require to spread services and annotations into separated folder with a deeper hierarchy or to move them on different places in the system. This approach was conducted to minimize time amount for scanning the application.

The framework also includes spring libraries for AOP and more, thus those developers who are not in favor of this enterprise technology might be discouraged in using them.

Chapter 6
Testing

This chapter aims to describe in detail test strategy, scenarios, stress tests and unit tests. IoT Framework was gradually tested by unit tests which are available in the repository and by test techniques in the following sections of this chapter. Framework was tested with real devices as well and these tests produced stress and integration test result presented in the end of this chapter. The following chapter describes more in detail proof of concept which provided platform for conducting the mentioned tests.

6.1 Test strategy

Application under test is virtually speared into following modules for testing purposes:

- Creating cache
- Invoking methods
- Evaluating conditions

The main testing quality characteristic was set to be the perfect functionality. Test and departmental goals are shown in table 6.1.

ID	Test goals and departmental test goals
1	System creates cache correctly
1.1	System reads folders with IoT services and annotations
1.2	System cached scanned structure
2	Manage aspect invokes methods correctly
2.1	Manage aspect gets methods linked to the managed annotation
2.2	Manage aspect invokes methods from cache correctly
3	Conditions evaluated correctly
3.1	To be invoked methods are evaluated against conditions correctly

Table 6.1. Test goals

Testing requirements of application modules are set in a relationship with test goals in the figure 6.1. Caching, invoking and condition evaluating is inherently interconnected. If caching would fail, other virtual modules would not simple work. Goals are therefore prioritize also via their respective order.

The risk class for testing modules is defined in the figure 6.2. Event invoking and condition evaluation works together and therefore they were merged together in order to avoid test case duplicates.

Test levels and test types are defined based on risk class in the figure 6.3 based on impacts when functionality is not working properly and probability of defect in the module. Defect probability is based on programming techniques used when module was coded.

Quality characteristic	Test goal	Process	Requirements	System module
Perfect functionality	System creates cache correctly	Caching	Cache is crated while starting the application	Cache module
	Manage aspect invokes methods correctly	Invoking	Invoking @Managed method triggers invoking of methods annotated by managed annotation.	Invoke module
	Conditions evaluated correctly	Invoking	Value in conditional annotation is evaluated with the application value.	Invoke module

Figure 6.1. Test prioritization

Process	Requirement	Damage	Damage explanation	Module	Defect probability	Defect probability explanation	Risk class
Caching	Links between methods and annotations that implement the framework, are correctly registered in the cache.	High	Expected functionality is not provided, thus the framework is unreliable.	Cache module	High	Caching uses reflection and thus is prone to make mistakes in scanned paths.	A
Invoking	Methods are invoked on correct objects with right values based on mapping from the cache.	High	Expected functionality is not provided, thus framework is unreliable.	Invoke module	Low	Programming paradigms are simple since using traditional POJO objects.	B

Figure 6.2. Test Risk Class

Quality characteristic: Perfect functionality Module	Risk class	Revision	Test levels Developer tests	System tests	Test types
Caching	A	Yes	Medium	High	Pairwise tests
Invoking	B	Yes	Medium	Medium	Pairwise tests

Figure 6.3. Test Levels and Test Types

6.2 Test scenarios

Let be the ConditionValue the integer value in the Condition Annotation. The ConditionValue is fixed for all use cases and ApplicationValues and Condition Annotations vary.

Invoking module was tested based on test strategy defined in the previous chapter and pairwise testing technique was chosen. Testing included 3 types of ApplicationValue, 3 mock IoT annotations, 5 mock methods and together 9 tests.

Table 6.2 shows distribution of Condition Annotations and Manage annotations. MA stands for Mock Annotation. For instance method `A` has two annotations: `@ValueEquals`, `@MockAnnotation`.

Following table 6.4 represents results of test cases. Each cell represents one test case. There were 3 additional methods (X, Y, Z) added to the previous set of methods A-E.

Method	Condition	MA1	MA2	MA3
A	ValueEquals	x		
B	ValueGreaterThan		x	
C	ValueGreaterThanOrEqual			x
D	ValueLessThan	x	x	
E	ValueLessThanOrEqual	x	x	x

Table 6.2. Invoking Test Annotation distribution

X, Y, Z were annotated by `@Manage` annotation with parameter shown in table 6.3. These methods were invoked with ApplicationValue 54, 55, 56 gradually and invoked respective methods stated in each table cell.

Method	Argument
X	`@MockAnnotation1.class`
Y	`@MockAnnotation2.class`
Z	`@MockAnnotation3.class`

Table 6.3. Distribution of arguments by @Manage annotated methods

For instance, annotation `@Manage` with argument `@MockAnnotation1.class` is carried by method X as shown in 6.3 and triggered with ApplicationValue 54 as an argument than invokes methods D and E.

ApplicationValue	X	Y	Z
54	D E	D E	E
55	A E	E	C E
56	..	B	C

Table 6.4. Invoking Module Test Cases

Caching module was tested with technique Pairwise testing coverage and was conducted 3 test cases with 5 mock IoT annotations, 3 application values and 1 ConditionValue. Table 6.5 is shown distribution of condition and invoking annotations, covering all possible distribution of annotations. Manage and IoT annotation are distributed to methods A-E. Note that methods B and D has also `@Manage` annotation. This fact will be used when creating use cases.

Method	Condition annotation	Invoking annotations
A	ValueEquals	`@MockAnnotation1`
B	ValueGreaterThan	`@Manage`, `@MockAnnotation1`
C	ValueGreaterThanOrEqual	`@MockAnnotation3`
D	ValueLessThan	`@Manage`, `@MockAnnotation3`, `@MockAnnotation1`, `@MockAnnotation2`
E	ValueLessThanOrEqual	`@MockAnnotation1`, `@MockAnnotation2`, `@MockAnnotation3`

Table 6.5. Caching Annotation Distribution

Test	Application Value	Methods
1	54	A, **D,** C, **E, E**
2	55	B, D, **E**
3	56	**B**, D, E

Table 6.6. Caching Module Test Cases

Method having annotations `@Manage` and `@MockAnnotation2` was triggered as origin event. ConditionValue was fixed on value 55. ApplicationValue varies as shown in table 6.6.

Results were as expected but minor changes and improvements were done in order to provide more cleaner and stable code.

6.3 Stress tests

This chapter aims to provide sufficient evidence of framework's capability in processing data on high scale. To be compared with an counterpart, it was setup an demo server implementing the very same logic but without using IoT Framework. The demo server used traditional approach with bean dependency.

At first, stress tests were conducted with IoT sensors, which are described more in detail in chapter 7. These test could not provide any comparison for gaps in communication over local network could not provide sufficient simulation of high system demands. Therefore a local python server was setup implementing TCP client with a configurable thread pool. The server sends in each thread ten HTTP requests synchronously. The values measured during testing are shown in table 6.7. First column indicates number of threads in thread pool. The second and the third column represents average times to process one request for server implementing the IoT Framework and without the framework respectively.

Threads	IoT Framework	No IoT Framework
1	27,3	24,3
2	14,05	13,95
5	6,96	6,2
10	3,45	4,46
20	2,945	2,495
30	2,533333	2,023333
40	2,6275	1,9025
50	2,49	2,062
100	3,851	2,065
200	2,936	1,6895
300	3,509	2,452333
350	3,946571	2,799143
380		2,706053
400		

Table 6.7. Stress Test Table

The graph of stress tests shown in figure 6.4 is projection of data in the table 6.7. Via this visualization is evident that IoT Framework is indeed competitive to implementation using spring bean dependencies.

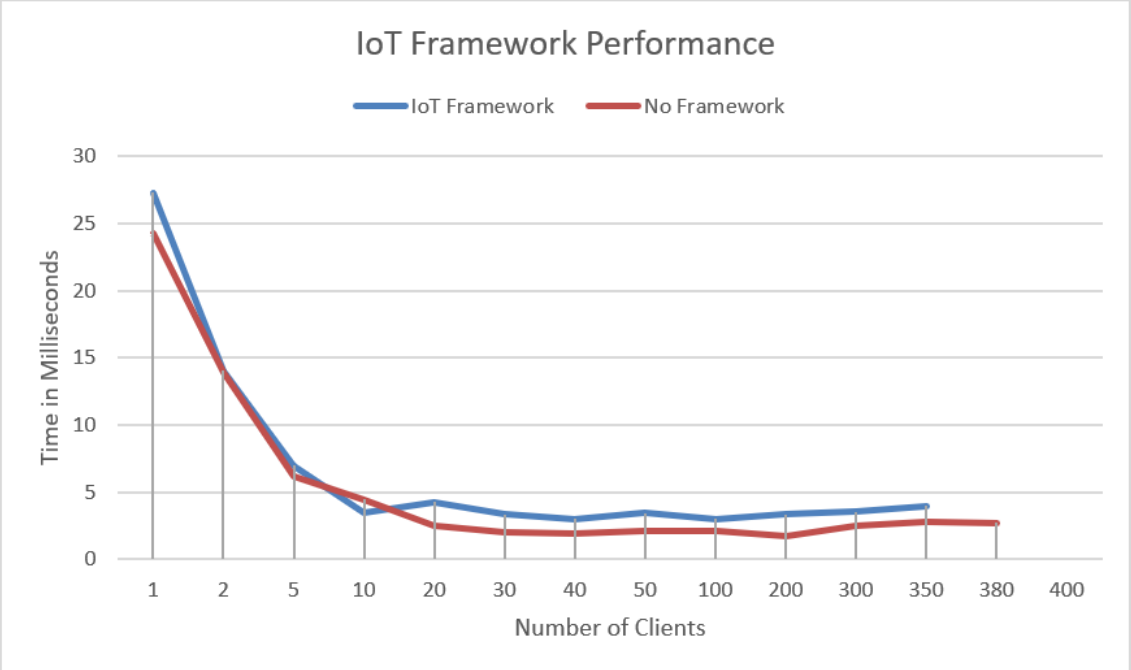

Figure 6.4. Stress tests

6.4 Unit tests

The framework was tested with mock annotations and methods to prove the correct behavior when creating cache, invoking methods from the cache and their filtering, as shown in the listing 6.1.

```
@Test
@DisplayName("Should Call Method")
void shouldCallMethod(){
    assertEquals(
      compareAnnotationService.shouldCallMethod(
          this.getMethodAnnotatedBy3GreaterThanOrEqual22(),
          new ApplicationValue(12)
      ),
      true,
     "Method called"
   );
}
```

Listing 6.1. Unit Test Showcase

United tests were conducted in order to tests all the elements of the system. The table 6.8 proves that framework's behavior is correct on the level of methods.

Method	Input(s)	Result(s)
Annotate the spring's onStartup method with IoTFramework annotation	Strings of path to custom services and annotations	Cache was created
Annotated custom annotation with @IoT	None	Custom annotation was added to the cache
Annotated a method with @Manage	Annotation to be managed	Manage annotation was not added to the cache
Annotated method with custom IoT annotation	None	Method and the link to the actual annotation was created in the cache
Annotated method with custom IoT annotation and @Compare	Conditional value	Custom method and compare annotation was added to the cache
Call method with @Manage annotation	None	Subsequent methods with managed annotation called
Call method with custom IoT annotation and @Compare IoT annotation and @Compare	Application Value	Conditional and application value was compared, and method invoked

Table 6.8. Unit Tests

Chapter 7
Use Case Study

The IoT Framework has an ambition to become indeed a handy tool for all kinds of projects. This requires not only to conduct testing properly, but also to assemble functioning proof of concept. It means to simulate the environment and devices in the most accurate way to show that the concept has reasonable changes to survive in real world. Framework will be used within the server application which is controlling set of devices which are sending data back and forth while expecting valid outputs and reactions at the same time. Table 7.1 shows devices that was used:

Device	Type	OS	Env	Application
PC	Acer M3800	Windows 10	JDK	IoT Server
SmartPhone	Samsung A3	Android 5.1.	NodeJS	Connector
Microcomputer	RaspberryPi	Raspbian	Python	RPiServer

Table 7.1. Specification of devices used in PoC.

These devices communicated via local network with static IPs. Routing secured router ASUS v4t. Deployment of all devices demonstrating proof of concept is shown at 7.1.

7.1 Raspberry Pi

Raspberry Pi was used to read data from thermometer and diode. It is very popular microcomputer which could be used in various context-aware applications and thus was very suitable for the proof of concept. Additionally, its CPU is enough powerful to conduct demanding stress tests and to read data from various sensors. Most current version of RaspberryPi3 B was chosen, it booted OS Raspbian from micro SD card Kingston 16 GB. Python 3.6 was downloaded alongside with a package manager pip. Virtual environment was created to provide basic infrastructure for development. IDE PyCharm from JetBrains was used to develop python scripts as it provided remote control via ssh.

```
def blink_led_diode():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(PIN,GPIO.LOW)
    time.sleep(1)
    GPIO.cleanup()
    return 2
```

Listing 7.1. Controlling Diode via GPIO pins

This proved to be more suitable for debugging than using Putty client and vim editor. Python scripts handled sensors via GPIO pins, which were connected to standard

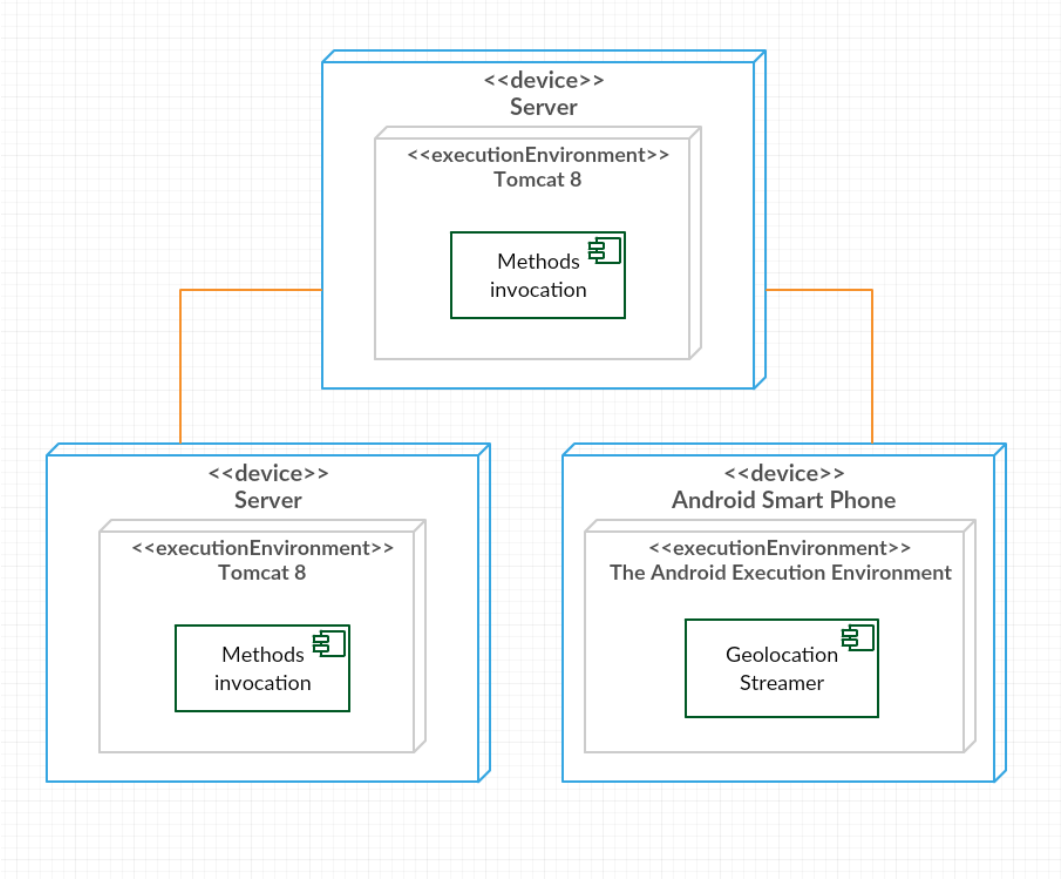

Figure 7.1. Deploy diagram of devices demonstrating proof of concept

breadboard with wired cables. DSCBC18 temperature sensor was used to read current temperature data, which proved to be reliable, accurate and its reaction to temperature changes appeared to be immediate. Red diode was controlled directed with build in GPIO libraries. 15K ohm resistors were used to reduce current flow. Sensors control scripts that provides following functionality:

- Raspberry Pi streams data from thermometer sensor to the server
- Raspberry Pi switches on the led diode for some interval
- Raspberry Pi reads data from the thermometer sensor and sends it to the requesting client

7.2 Smart Phone Application

Smart Phone Application was created to send data periodically to the desired location. It includes various of sensors, which provides complex data of the users environment and thus it was chosen as a device for proof of concept. Device Samsung A3 2015 with OS Android 5.1. was used as a platform. Application was developed on PC while using Ionic framework based on Angular2.

```
sendDataPeriodically(){
    this.geolocationProvider
  .sendGeolocation(this.ipv4, this.position);
    this.sendingInProgress = true;
    IntervalObservable.create(this.interval)
```

```
        .takeWhile(() => this.sendingInProgress)
        .subscribe(() => {
          this.geolocationProvider
    .sendGeolocation(this.ipv4, this.position, true);
        });
    }
```

Listing 7.2. Sending Geolocation to Server

This required to install ionic framework via npm package manager. Development was conducted in WebStorm IDE from JetBrains. Cordova from Apache was used to deploy application with APK editor. Application provides following functionality which was represented by the UI as shown on the picture 7.2:

- send geographic coordinates once, mainly for debug reasons
- send geographic coordinates repeatedly
- notify user about the result of data sending

7.3 Server application

Server application implements the IoTFramework and shows the functionality of the framework working with real devices. Spring Boot was chosen for reasons mentioned in previous chapters. Framework was included via project management tool Maven. IDE IntelliJ IDEA was used for server application development.

7.4 Proof of Concept Use Cases

Raspberry Pi starts sending data from temperature sensor in two second interval via HTTP requests. Server application receives data on the REST API and transmits data to services implementing the framework. If the temperature is above 26 degrees Celsius, server sends request to switch the diode on and the event is notified. If the temperature equals 25 degrees Celsius, request to blink the diode twice is sent. If the temperature falls under 25, the diode is switched off. Sensor connection schema together with Raspberry Pi is shown on the picture 7.3. Smart Phone Application sends geographic coordinates, if the latitude value falls under 49.228866, it requests the temperature data and notifies user about current temperature and geographic coordinates.

7.5 Usage in large scale

The IoT Framework can be used in large scale processing hundreds different types of sensors each of them in thousands of pieces and together building dozens of use cases with complicated conditions. This can be found in industry, avionics, factories or hospitals. Managers and architects in charge of business aspects of projects have following reasons to use this technology.

Spring applications are used mainly in large scale and therefore any component, especially framework must prove its capability in large scale. This was proved in chapter 6, primarily in stress tests. Simulation of up to 400 of sensors generating together up to 4000 requests in 10 seconds showed framework's effectiveness and capability to manage large amount of data in short period of time and proved to be competitive against Spring dependency.

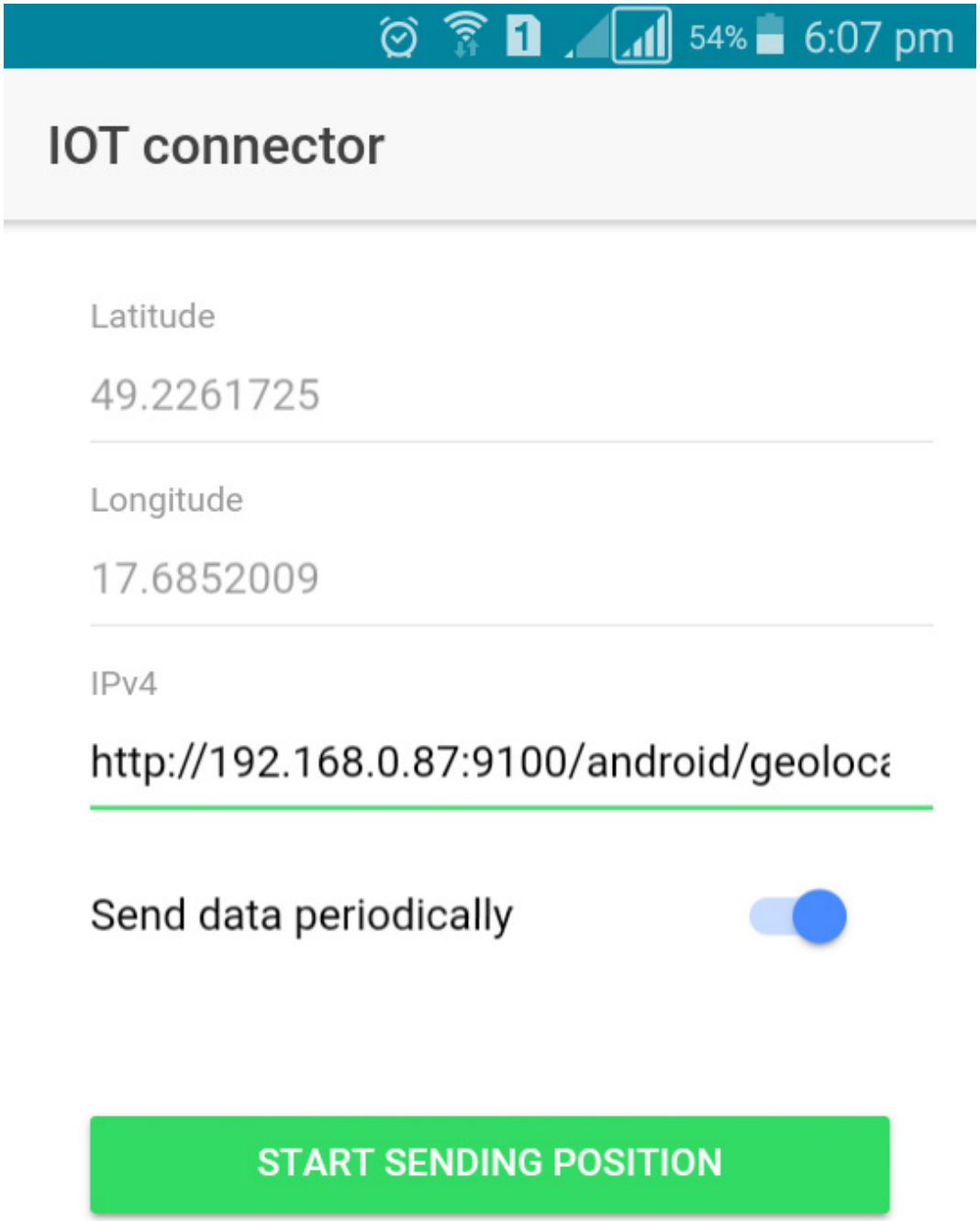

Figure 7.2. Geolocation Android Application

The IoT Framework can be used in all application where is high cohesion among components and services. In such application there is very often circular dependencies, but such bean dependencies are not allowed to use in application. But the framework indeed solve this problem when introducing scanning through application via reflection and storing the connections in N:M relationships.

Connection as a service was a principle described in the chapter 1. IoT Framework proves to be enough sufficient for this pattern as it provides indeed smooth and orderly

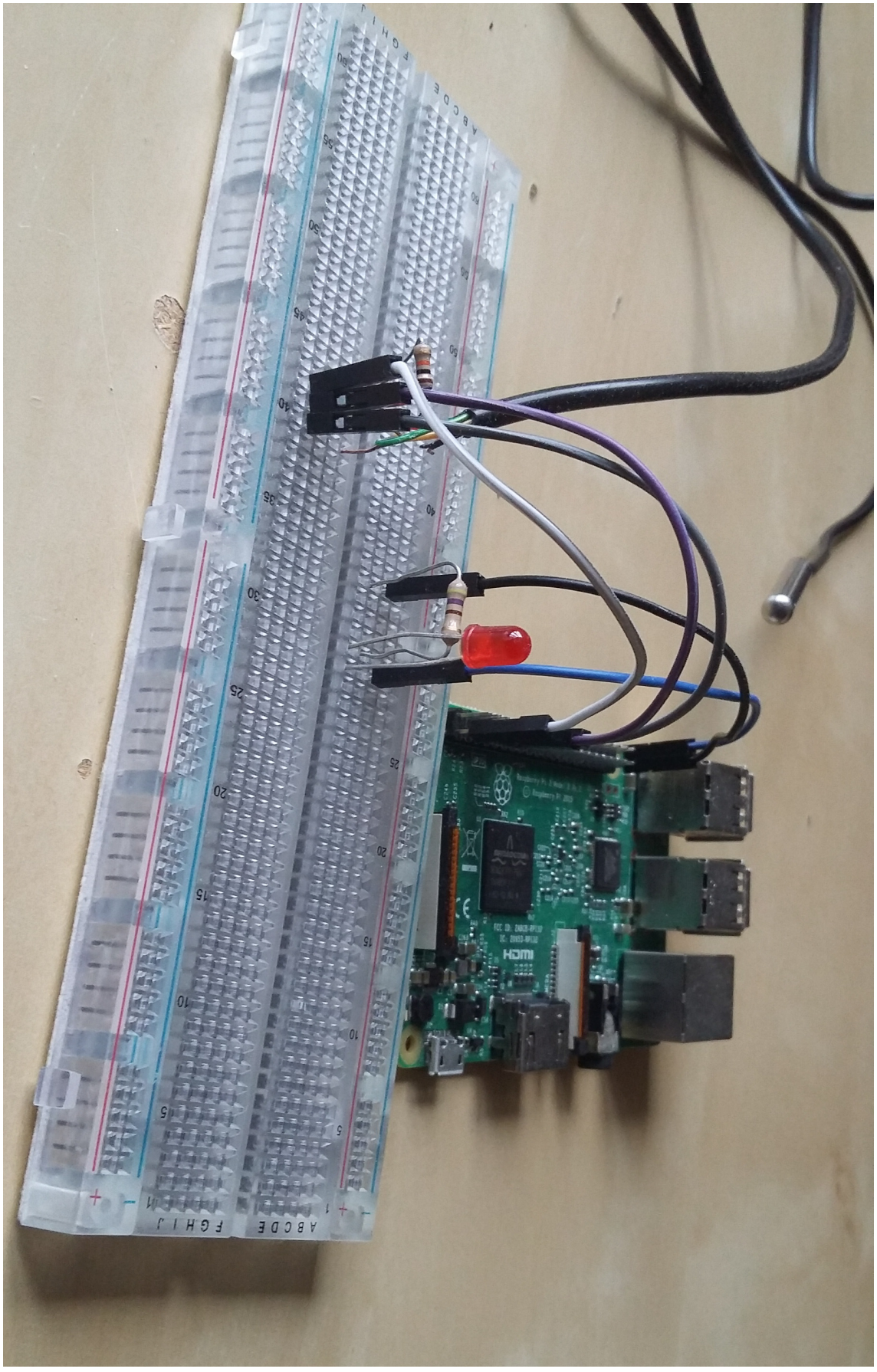

Figure 7.3. Sensor Connection Schema with Raspberry Pi

way of scaling applications as new sensor types might be added to the system and at the same time remains equally efficient when the number of sensors is increasing over time.

7.6 Usage in context-aware applications

Context-aware applications have many special requirements not only on types of sensors and overall hardware structure but also to programming paradigms. The nature of context aware applications is changing its state based on inputs form outside environment. Event driven programming is key answer on incoming data and its processing. Framework implements this paradigm when introducing inner cache and let control of invoking and caching purely on framework. This is would be appreciated by developers as debugging and having clear overview of developing systems becomes indeed sustainable as applications grows.

Testability is another strong reason for system architects to choose the framework for IoT solutions. Unit testing of methods invoking and condition evaluating is from programming point of view once tested when framework was crated. Testers only needs to evaluate the validity from business point of view and do not have to go further in technical details in autowired dependencies and coding patterns.

7.7 Benefits for developers

Framework is distributed as executable jar, thus it is highly manageable by software project management and comprehension tools as Maven or Gradle. Moreover, Maven repositories offers option to place own jar's to public repositories and allow anyone to include framework in their projects. Quick and straightforward integration within both more complicated and simple projects enables rapid frameworks usages. Framework consist only of 15 classes taking up about 5Kb in jar. Stress tests in chapter 6 only confirmed its competences.

Framework enables programmer to transfer business requirements directly into the code without needing to sketch diagrams or debug complicated if statements as shown in chapter 4.

In the modeling proof of concept in this chapter it comes to the system a request with a value from a thermometer, based on this request event of setting on the diode, sending the notification and displaying of temperature should be triggered. In the standard procedure, the programmer would have to inject into TemperatureService following beans: DiodeService, NotificationService and others. A higher number of service classes and their interconnection would result in the situation in which each service class has injected all the other service classes. This is difficult for programming, when many parts of code are repeated, the code is cluttered and we are straining the spring container that cares about the beans dependency.

The concept of annotations as meta data holders has been used since 1.5 JDK and JAVA programmers have become aware to this technique even more when aspect oriented programming has become more popular. The framework's setup is simple and requires only to place 'IoTFramework' annotation to the method, which is triggered at the very end of application startup. Next, programmer use only 'IoT' annotation to annotated custom annotation used to manage event driven method triggering. More over, current smart IDE's enables overview of methods of certain class, thus programmer can quickly manage methods and its annotations in smart views. This all shows

how user friendly framework is and thus it is expected programmers get used to the framework's features quickly.

Frameworks development took in mind fast growing community of developers, who shares their knowledge and want to contribute to the public projects. New ideas and improvements are welcomed, thus code was structured and coded according to clean code pricples to let anyone to contribute. This is especially the case in terms of security since framework can be easily extended by role and permission management.

Chapter 8
Conclusion

The main task of this bachelor thesis was to design and implement the framework for the use of IoT sensor data in context-aware applications. Design and implementation was based on theoretical background presented in chapter 3. Existing solutions and current challenges defined scope in which the framework would bring significant benefits for developers.

The framework was created with a consideration of three important aspects of context-awareness. It focuses mainly on optimizing the automatic performance of services, unified presentation of information and services to the user, and design of context retention as information for further use. By setting several programming techniques, it responds to one of the major challenges of context-aware programming, namely the discovery of context-aware applications that would help users in interacting with computing services. Additionally, using defined procedures helps to improve the infrastructure to support the implementation of context-aware applications.

The research resulted in framework that is better or competitive compare to traditional approach in following aspects: Boilerplate Code, Lines of code, Run-time Code Execution (competitive), Code Sustainability and Code Reuse.

These benefits were tested and proofed via stress tests 6, which showed framework's capability in handling large amount of client simultaneously in time competitive to native spring dependency injection. Code comparison 5 or traditional and IoT Framework approach showed that developers would need three time less code and programming advanced tasks goes in clear and logical way. The framework was proofed to work with a real devices since the framework was integrated into the server application which worked together with Android Application, Python application in Raspberry Pi, temperature sensor and diode 7. Therefore it is expected the framework to be capable of usage in more general use cases as showed in chapter 4.

Testing and use case study proofed that framework met all requirements and design qualities in sufficient way and therefore it might be presumed that all assigned tasks were solved successfully.

Future development of the framework can be conducted in the domain of security and further optimization. Security is an challenging issue in IoT world and context-aware application as well. Event execution would be preceded by authentication of the client side and further use of different sorts of application functionality would be allowed to user with certain permissions. Optimization of reflection, cache creation and event invocation might result in even better performance. Current solution also force developers to place annotations and methods implementing the framework into two folders, but their usage might be required through the application.

References

[1] "Chin-Lung Hsu, Hsi-Peng Lu, and Huei-Hsia Hsu". "Adoption of the mobile Internet: An empirical study of multimedia message service (MMS)". *"Omega"*. "2007", "35" ("6"), "715 - 726". DOI "https://doi.org/10.1016/j.omega.2006.03.005". "Special Issue on Telecommunications Applications".

[2] "Luigi Atzori, Antonio Iera, and Giacomo Morabito". "The Internet of Things: A survey". *"Computer Networks"*. "2010", "54" ("15"), "2787 - 2805".

[3] Gartner Inc. "Hype Cycle for the Internet of Things. "2017",

[4] "Abowd. In: *"Handheld and Ubiquitous Computing: First International Symposium*. "Berlin: "Springer Berlin Heidelberg", ISBN "978-3-540-48157-7". "https://doi.org/10.1007/3-540-48157-5_29" .

[5] Boris Golubovic. *IoT: Next level reducing total cost of ownership*. 2017. mbtmag.com/article/2017/02/iot-next-level-reducing-total-cost-ownership.

[6] Y. Ni, C. L. Xing, and K. Zhang. Connectivity as a Service: Outsourcing Enterprise Connectivity over Cloud Computing Environment. 2011, 1-7. DOI 10.1109/CAMAN.2011.5778899.

[7] *AWS IoT*. https://aws.amazon.com/iot/. 2017.

[8] *Azure IoT Suite*. https://www.microsoft.com/en-us/internet-of-things/azure-iot-suite. 2017.

[9] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys Tutorials*. 2014, 16 (1), 414-454. DOI 10.1109/SURV.2013.042313.00197.

[10] Manfred Sneps-Sneppe Dmitry Namiot. On Micro-services Architecture. *International Journal of Open Information Technologies*. 2014, 2 (9), 24-27. DOI 10.1109/SURV.2013.042313.00197.

[11] "Abu-Elkheir. "Data Management for the Internet of Things: Design Primitives and Solution". *"Sensors (Basel)"*. "2013", "54" ("13"),

[12] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications. *Hum.-Comput. Interact.*. 2001, 16 (2), 97–166.

[13] F. Khodadadi, A. V. Dastjerdi, and R. Buyya. Simurgh: A framework for effective discovery, programming, and integration of services exposed in IoT. 2015, 1-6. DOI 10.1109/RIOT.2015.7104910.

[14] M. Kranz, P. Holleis, and A. Schmidt. Embedded Interaction: Interacting with the Internet of Things. *IEEE Internet Computing*. 2010, 14 (2), 46-53. DOI 10.1109/MIC.2009.141.

[15] *Spring*. 2017. docs.spring.io/spring/docs/current/spring-framework-reference/overview.html.

[16] *Spring AOP*. 2017.
`docs.spring.io/spring/docs/current/spring-framework-reference/core.html`.
[17] *Maven*. https://maven.apache.org. 2017.
[18] *Tomcat*. http://tomcat.apache.org. 2017.

Appendix A
List of Abbreviations

AOP	Aspect oriented programming
MVC	Model View Controller
OOP	Object oriented programming
IoT	Internet of Things
TCO	Total Cost of Ownership
GPS	Global Positioning System
M2M	Machine to machine
API	Application Programming Interface
JEE	Java Platform, Enterprise Edition
Spring	Spring Boot

A.1 Terms

IoT Annotation	The annotation is used for annotating annotations participating in chain event invocation
Custom Annotation	The annotation created by the developer and annotated with IoT Annotation
ApplicationValue	Object for storing value from the sensor, the value is compared against value in condition annotation
Condition Annotation	The annotation holds statically defined value which is compared with the methods argument, Application Value, in the run-time of the method
ConditionValue	Value defined in the Condition Annotation, compared on the runtime with application value in the argument of the method holding the Condition Annotation
Manage Annotation	Enables invoking of all methods annotated by annotation in the argument of the manage annotation
Manage aspect	With framework's services conducts chain event invocation
Origin event	Method annotated by Manage Annotation
Subsequent event	Method annotated with the Custom Annotation or Annotations and optionally with Condition Annotation
IoT framework annoation	The annotation provides definition of packages with annotations with IoT Annotation and methods with Manage Annotations and Condition Annotations

Appendix B
Listings

Appendix C
Content of attached CD

The CD content is organized into directories and files as follows:

- iot-framework-thesis.pdf - the bachelor thesis
- bachelor-thesis-resources - TEX sources
- bachelor-thesis-resources/pages - bachelor thesis chapters in separeted TEX files
- bachelor-thesis-resources/assets - Figures used in the thesis
- bachelor-thesis-resources/resources - Schemes, sheets, diagrams, etc.
- iot-framework.zip - Java Sources of IoT Framework
- iot-framework.jar - IoT Framework Executable .jar
- android-application.zip - Android Geolocation Streamer Application source code and .apk
- rpi-scripts.zip - Python 3 scripts for sensor data processing
- no-framework-server.zip - Spring Boot server with conventional approach of implementing simple context-aware application
- iot-framework-server.zip - Spring Boot server using IoT Framework for implementing the same context-aware application
- stress-test-scripts.zip - Tests written in Python3 conduction stress testing of the server with and without the framework.